Electronic Research Administration

NIH System to System (S2S) “Client” Developer Guide

Version 1.1.0.0

TABLE OF CONTENTS

INTRODUCTION

Background

This document provides software developers with programming level details for the eRA Exchange System-to-System Client (S2Sclient) software package. The aim is to allow developers to quickly adapt the S2Sclient for their own local application needs. S2Sclient is written in Java and so developers must be already familiar with creating Java software solutions.

The S2Sclient package is provided as a convenience to allow implementers to rapidly deploy a baseline system and test that it operates correctly directly with the NIH S2S exchange server. Implementers are responsible for configuring and adapting this baseline package to operate in their own actual environments. Basic installation and setup instructions are provided for a typical envisioned Windows server system. For details on installing and configuring the S2Sclient see the separate installation guide document.

The S2Sclient provides two mechanisms for handling inbound and outbound transactions directly with the NIH S2S exchange server environment. One mechanism is the delivery sub-directory structure and the other is data handlers. The S2Sclient provided comes with data handlers for each of the default message types. These can be replaced or extended and adapted and the S2Sclient configuration setup to point to alternate data handlers. This guide gives instructions on adapting and replacing data handlers and using the delivery sub-directory structure.

The sub-directory mechanism includes automated delivery and receipt handling – this allows manual testing of payload exchanges, as well as providing a simple integration mechanism. The data handlers meanwhile provide for automated payload handling services implementation.

This document is intended to supplement the main documentation from Hermes and to provide a quick start for the code samples provided with the S2Sclient. Developers seeking more in-depth information should also reference the Hermes project documentation as well (copies of these additional PDF files are included in the installation package under a \docs directory).

The package includes sample interfacing mechanisms and test messages to establish that the S2Sclient is operating correctly with the eRA NIH server systems in both the Ext-UAT testing area and the production environment.

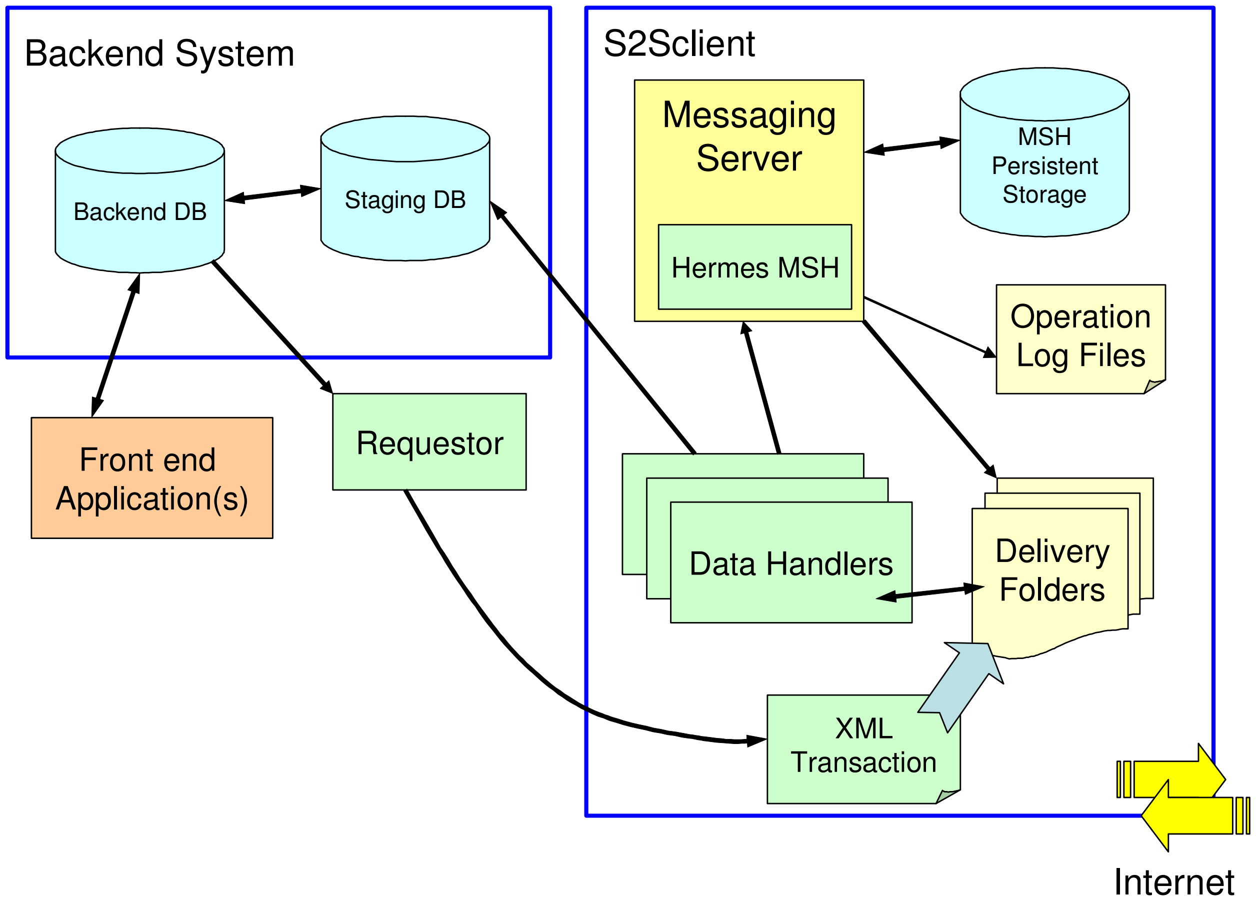
The baseline functionality will be adapted and extended by eRA NIH and new S2Sclient packages will then be made available with those new features and capabilities as required to support the grants application processing.

Planned transactions include: Status request, Person information request, Person information update, Validations service, Grant Verification image, and Notice of Grant Award transaction.

Normative XML W3C XSD-schema structures are provided that defines the layout and basic content model for each transaction type. These schemas will be updated and added to as necessary. Also sample messages are provided that conform the to schema layout structures.

The interchange model is shown below in Figure 1.

Figure 1 – Schematic of interchange models using the S2Sclient system.



Incoming transactions are processed by matching data handler based on message type indicated in the message envelope header and placed in the appropriate incoming folder.

Outgoing transactions are placed in the delivery folder for that message type; the data handler listener service then picks up that transaction, packages it as a valid outbound message, then passes it to the MSH for delivery and re-locates the original transaction to the corresponding archive.

Prerequisites

Development System

The following is a summary of the development environment required to allow developers to extend and modify the default S2Sclient package.

- Default environment:
    - Operating Systems: Windows 2000/XP
    - Application Servers: Tomcat 4.x is the default provided in the install package.
    - Database Engines: DerbyDB is the default embedded SQL database used.
    - Java Versions: 1.4.2 - Development Kit (SDK) version only (later versions are currently not supported and do not work with the package as configured)
    - Java software development environment to allow compilation and testing of modified Java classes and programs.

This is the default environment that is installed as is. All the components such as Tomcat and Derby are pre-configured. However this packaging is not the sole deployment environment. Equivalent components may be substituted by implementers, based on their own local system operational requirements. Examples include using different servlet services instead of Tomcat, or different SQL compatible databases such as Oracle. Implementers are responsible for configuring these changes themselves, however a wide variety of commonly available COTS solutions are known to be compatible.

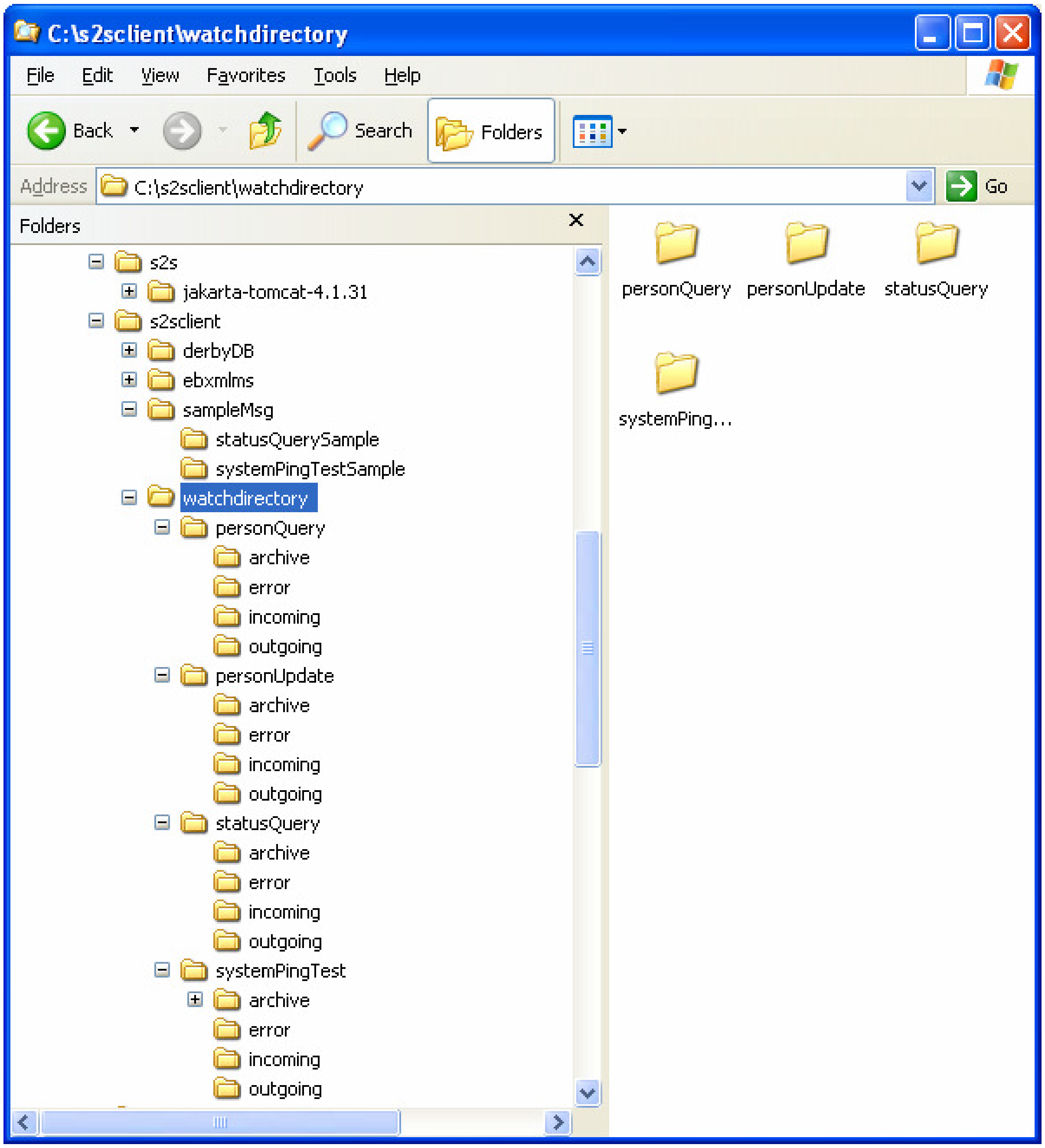
Java Environment

You will need to download and install the Java 1.4.x SDK system from Sun Microsystems (http://java.sun.com/j2se/1.4.2/download.html). The JDK development environment must be present to run the server environment (the JRE alone will ***not*** work), also newer JDKs will not work because of the older version of TomCat is included in this NIH distribution. (Note: We expect to upgrade at some point once we have completed baseline system verification).

See the separate setup and installation guide for details on configuring the Java environment correctly.

Overview of Delivery Structure Mechanism

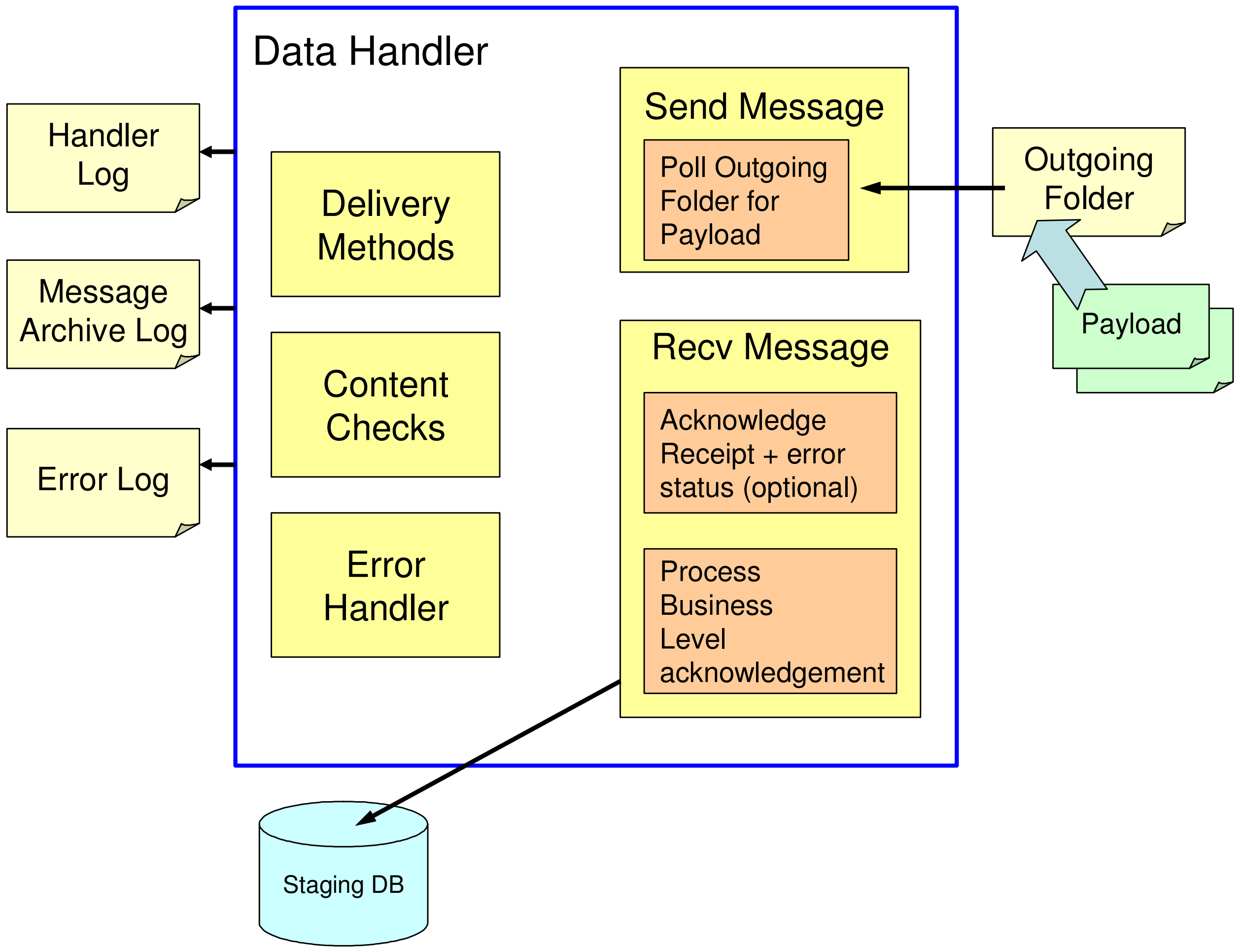
The main S2Sclient is configured to use a delivery sub-directory structure for outbound and inbound transaction handling. Each message type has its own sub-directory branch.



The Data Handler Mechanism with Hermes

This figure shows the overall components within the data handler.

Figure 2 – Data Handler



Included here in this document is sample source code for a default data handler.   We now discuss the source code and highlight those areas that can be configured to provide integration with data stores and to implement business rules as required.

First we review the source code for the sample data handler and look at an example of simple processing. In this case the code belongs to the class that simply copies the inbound transaction into a file in the matching incoming folder directory for that message type.

The source code of the DefaultClientProcessor is listed below:

```
package gov.nih.cgaprefimpl.processor;

import gov.nih.cgaprefimpl.*;
import gov.nih.cgaprefimpl.stageddelivery.DefaultPayloadService;
import java.io.*;
import java.util.*;
import javax.activation.DataHandler;
import javax.activation.DataSource;

/**
 * This class is an example of a data handler.
*/

public class DefaultClientProcessor extends DefaultPayloadService {

  public void processPayloadReceipt(final MessageVO messageVO)
  throws Exception {
     log.info("Processing payload Receipt: " + messageVO);

  try {
     // store the payloads
     final Map payloadMap = messageVO.getPayloadMap();
     for (final Iterator payloadIterator = payloadMap.keySet().iterator();
                payloadIterator.hasNext();) {
          final String contentId = (String) payloadIterator.next();
          log.info("PayLoad Content Id: "  + contentId);

          final String absolutePath =
                   storePayload(contentId, payloadMap.get(contentId), messageVO.getCpaID(),
                                messageVO.getConversationID(), messageVO.getAction());

          log.info("Stored payload in: "  + absolutePath);

           String conversationID = messageVO.getConversationID();
           int i = conversationID.indexOf("_");
           String messageType = conversationID.substring(0,i);
           cat.debug("Message Type "+messageType);

           if(messageType.equals(S2SConstants.STATUS_QUERY))
                        {
                               StatusMessageProcessor st = new StatusMessageProcessor();
                               st.process(messageVO,messageType);
                        }else if(messageType.equals(S2SConstants.PERSON_QUERY))
                        {
                               PersonInfoMessageProcessor pi = new PersonInfoMessageProcessor();
                               pi.process(messageVO,messageType);
                        }else if(messageType.equals(S2SConstants.PERSON_UPDATE))
                        {
                               PersonUpdateProcessor pi = new PersonUpdateProcessor();
                               pi.process(messageVO,messageType);
                        }


     } // end for

    } catch (Exception e) {
       // send the error status
          log.error(e);
    } // end try catch
  } // end processPayloadReceipt(MessageVO)


  protected String storePayload(final String contentId,
                                  final Object payloadObject, final String fromCPA,
                                  final String conversationID, final String messageType)
                                  throws Exception {

           // store the payloadObject
                        } catch (Exception e) {
                                  throw e;
                        }
                        return file.getAbsolutePath();
            } // end storePayload
}
```

The code first takes the incoming transaction and stores it into the appropriate received sub-directory. After that validation or other processing is performed depending on the type of incoming transaction identified by calling a specific message processor class. If an error occurs it is logged accordingly and then the process returns.

Next we consider details of configuring the S2Sclient to use a specific message processor. In this example there is first a main class that determines the type of transaction received and then calls the matching message processor.

The message processor can then perform checking content values, referencing a backend database and generating response messages and updating database tracking tables as needed.

Following this we next detail how to configure the S2Sclient to accept a new message processor class for deployment.

Configuring the Data Handlers for Deployment

For deployment purposes the data handler class needs to access the MessageVO and DefaultPayloadService classes and be configured in the appropriate mhs ***processoragents.properties*** parameter file so that the ExchangeClient startup process will invoke the new class. In the case of TomCat this class jar file most be located in the search path.

The default content of the parameter file is:

```
systemPingTestResponseToSP.class=gov.nih.cgaprefimpl.processor.DefaultClientProcessor
systemPingTestResponseErrToSP.class=gov.nih.cgaprefimpl.processor.DefaultClientProcessor

statusQueryResponseToSP.class=gov.nih.cgaprefimpl.processor.DefaultClientProcessor
statusQueryResponseErrToSP.class=gov.nih.cgaprefimpl.processor.DefaultClientProcessor

personQueryResponseToSP.class=gov.nih.cgaprefimpl.processor.DefaultClientProcessor
personQueryResponseErrToSP.class=gov.nih.cgaprefimpl.processor.DefaultClientProcessor

personUpdateResponseToSP.class=gov.nih.cgaprefimpl.processor.DefaultClientProcessor
personUpdateResponseErrToSP.class=gov.nih.cgaprefimpl.processor.DefaultClientProcessor
```

Each message type has its own mapping to invoke a class. Currently all the incoming messages invokes the default class
`“gov.nih.cgaprefimpl.processor.DefaultClientProcessor”.`

This can be replaced with any other class. The new class has to be available in Tomcat (or whichever app server is being used) classpath location directory.

The next section looks at actual schema structures for the transactions.

Schema and Transaction Notes

The following schemas are currently implemented and their associated transactions:

- personinforequest
- personinforesponse
- personinfoupdaterequest

- status_schema

In addition these schemas provide common definitions:

- commontypes
- nihschema
- rarschema

The documentation for S2S transactions is provided in separate documents that detail the actual interchanges that are supported. (Note: The transactions and documentation are under development.)

Business Process Interactions

The following table shows the current business process interactions supported.

Process	Request Transaction	Response
Person Information Request	personinforequest	personinforesponse
Person Information Update	personinfoupdaterequest	personinforesponse; with pires:RequestID = “UPDATE”
Application Status Request	status_schema – request elements	status_schema – with response elements completed

Each request is followed by a matching response transaction, or an error transaction.

Configuration Tables

The S2Sclient data handlers are controlled by various configuration tables – these are itemized here. Advanced users may manually edit these tables as needed to change the setup and operations of the S2Sclient.

Filename	Location	Description
Web.xml	..\jakarta-tomcat-4.1.31\webapps\refimpl\WEB-INF\	CGAP startup servlet

exchangeclient.properties	..\jakarta-tomcat-4.1.31\webapps\refimpl\WEB-INF\classes\	#CPA File Location
		cpa.file.location=c:/S2SClient/cpa.xml
		
		# What directory are the files Picked up from
		watch.directory=c:/S2SClient/watchdirectory
		
		# What is the interval in which to check that directory (milliseconds)
		polling.interval=5000
		
		# Destination CPA - hard wired for LRP CPA
		destination.cpa=LRPCPA
		
		supported.message.types=
		systemPingTest, statusQuery, personQuery, personUpdate

messageHandlerFactory.properties		Associates the transaction type with the handler class
processoragents.properties		Class / processor mapping for message handlers
startup.properties		Invokes startup handler classes
stageddelivery.properties		Specifies the staged delivery message handler (pulled message handling)